User's Guide for Quantum-ESPRESSO

(version 3.0)

Contents

1 Introduction

This guide covers the installation and usage of Quantum-ESPRESSO (opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization), version 3.0.

The Quantum-ESPRESSO package contains the following codes for the calculation of electronic-structure properties within Density-Functional Theory, using a Plane-Wave basis set and pseudopotentials:

- PWscf (Plane-Wave Self-Consistent Field).
- CP (Car-Parrinello).

and the following auxiliary codes:

- PWgui (Graphical User Interface for PWscf): a graphical interface for producing input data files for PWscf.
- atomic: a program for atomic calculations and generation of pseudopotentials.
- iotk: an Input-Output ToolKit.

The Quantum-ESPRESSO codes work on many different types of Unix machines, including parallel machines using Message Passing Interface (MPI). Running Quantum-ESPRESSO on Mac OS X and MS-Windows is also possible: see section 2, "Installation".

Further documentation, beyond what is provided in this guide, can be found in:

- the `Doc/` directory of the Quantum-ESPRESSO distribution

  In particular the `INPUT_*` files contain the detailed listing of available input variables and cards.
- the various `README` files found in the distribution
- the Pw_forum mailing list (`pw_forum@pwscf.org`)

  You can subscribe to this list and browse and search its archives from the PWscf web site (`http://www.pwscf.org/`). Only subscribed users can post. Please search the archives before posting: your question may have already been answered.
- the "Scientific Software" page of the Democritos web site (`http://www.democritos.it/scientific.php`)

This guide does *not* explain solid state physics and its computational methods. If you want to learn that, read a good textbook.

1.1 Codes

PWscf can currently perform the following kinds of calculations:

- ground-state energy and one-electron (Kohn-Sham) orbitals
- atomic forces, stresses, and structural optimization
- molecular dynamics on the ground-state Born-Oppenheimer surface, also with variable-cell
- Nudged Elastic Band (NEB) and Fourier String Method Dynamics (SMD) for energy barriers and reaction paths
- phonon frequencies and eigenvectors at a generic wave vector, using Density-Functional Perturbation Theory
- effective charges and dielectric tensors
- electron-phonon interaction coefficients for metals
- interatomic force constants in real space
- third-order anharmonic phonon lifetimes
- Infrared and Raman (nonresonant) cross section
- macroscopic polarization via Berry Phase

All of the above work for both insulators and metals, in any crystal structure, for many exchange-correlation functionals (including spin polarization and LDA+U), for both norm-conserving (Hamann-Schlüter-Chiang) pseudopotentials in separable form, and — with very few exceptions — for Ultrasoft (Vanderbilt) pseudopotentials. Non-colinear magnetism and spin-orbit interactions are also implemented. Finite electric fields are implemented in both the supercell and the "modern theory of polarization" approaches (the latter is still at an experimental stage). Various postprocessing and data analysis programs are available.

CP can currently perform the following kinds of calculations:

- Car-Parrinello molecular dynamics simulation
- geometry optimization by damped dynamics
- constant-temperature simulation with Nosè thermostats (including Nosè-Hoover chains for each atom)

- variable-cell (Parrinello-Rahman) dynamics
- Nudged Elastic Band (NEB) for energy barriers and reaction paths
- String Method Dynamics (in real space)
- dynamics with Wannier functions and under finite electric fields

Spin-polarized calculations. CP works with both norm-conserving and Ultrasoft pseudopotentials. There are implementations of a dynamics for metals using conjugate-gradient algorithms, and of the meta-GGA functionals. Both are at an experimental stage.

1.2 People

The maintenance and further development of the Quantum-ESPRESSO code is promoted by the DEMOCRITOS National Simulation Center of INFM (Italian institute for condensed matter physics) under the coordination of Paolo Giannozzi (Scuola Normale Superiore, Pisa), with the strong support of the CINECA National Supercomputing Center in Bologna under the responsibility of Carlo Cavazzoni. Currently active developers include Gerardo Ballabio (CINECA), Stefano Fabris, Adriano Mosca Conte, Carlo Sbraccia (SISSA, Trieste), Anton Kokalj (Jožef Stefan Institute, Ljubljana).

The PWscf package was originally developed by Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso (SISSA), Paolo Giannozzi, and others.

The CP code is the result of the merging of two codes: CP and FPMD, both based on the original code written by Roberto Car and Michele Parrinello. CP was developed by Alfredo Pasquarello (IRRMA, Lausanne), Kari Laasonen (Oulu), Andrea Trave (LLNL), Roberto Car (Princeton), Nicola Marzari (MIT), Paolo Giannozzi, and others. FPMD was developed by Carlo Cavazzoni, Gerardo Ballabio (CINECA), Sandro Scandolo (ICTP, Trieste), Guido Chiarotti (SISSA), Paolo Focher, and others.

PWgui was written by Anton Kokalj and is based on his GUIB concept (`http://www-k3.ijs.si/kokalj/guib/`).

The pseudopotential generation package "atomic" was written by Andrea Dal Corso and it is the result of many additions to the original code by Paolo Giannozzi.

The input/output toolkit "iotk" was written by Giovanni Bussi (S3, Modena).

An alphabetical list of further contributors includes: Dario Alfè, Francesco Antoniella, Mauro Boero, Nicola Bonini, Claudia Bungaro, Paolo Cazzato, Davide Ceresoli, Gabriele Cipriani, Matteo Cococcioni, Cesar Da Silva,

Alberto Debernardi, Gernot Deinzer, Oswaldo Dieguez, Andrea Ferretti, Guido Fratesi, Ralph Gebauer, Martin Hilgeman, Eyvaz Isaev, Yosuke Kanai, Axel Kohlmeyer, Konstantin Kudin, Michele Lazzeri, Kurt Maeder, Francesco Mauri, Nicolas Mounet, Pasquale Pavone, Mickael Profeta, Guido Roma, Manu Sharma, Alexander Smogunov, Kurt Stokbro, Pascal Thibaudeau, Antonio Tilocca, Paolo Umari, Renata Wentzcovitch, Yudong Wu, Xiaofei Wang, and let us apologize to everybody we have forgotten.

This guide was mostly written by Paolo Giannozzi, Gerardo Ballabio, Carlo Cavazzoni.

1.3 Contacts

The web site for Quantum-ESPRESSO is:

`http://www.quantum-espresso.org/`

Releases and patches of Quantum-ESPRESSO can be downloaded from this site or following the links contained in it.

Announcements about new versions of Quantum-ESPRESSO are available via a low-traffic mailing list Pw_users: (`pw_users@pwscf.org`). You can subscribe (but not post) to this list from the PWscf web site.

The recommended place where to ask questions about installation and usage of Quantum-ESPRESSO, and to report bugs, is the Pw_forum mailing list (`pw_forum@pwscf.org`). Here you can obtain help from the developers and many knowledgeable users. You can subscribe to this list and browse and search its archive from the PWscf web site. Only subscribed users can post Please search the archives before posting: your question may have already been answered.

If you specifically need to contact the developers of Quantum-ESPRESSO (and only them), write to `pwscf@pwscf.org`.

Other pointers:
DEMOCRITOS: `http://www.democritos.it/`
INFM: `http://www.infm.it/`
CINECA: `http://www.cineca.it/`
SISSA: `http://www.sissa.it/`

1.4 Terms of use

Quantum-ESPRESSO is free software, released under the GNU General Public License (`http://www.pwscf.org/License.txt`, or the file `License` in the distribution).

All trademarks mentioned in this guide belong to their respective owners.

We shall greatly appreciate if scientific work done using this code will contain an explicit acknowledgment and a reference to the Quantum-ESPRESSO web page. Our preferred form for the acknowledgment is the following:

> *Acknowledgments:*
> Calculations in this work have been done using the Quantum-ESPRESSO package [*ref*].
>
> *Bibliography:*
> [*ref*] S. Baroni, A. Dal Corso, S. de Gironcoli, P. Giannozzi, C. Cavazzoni, G. Ballabio, S. Scandolo, G. Chiarotti, P. Focher, A. Pasquarello, K. Laasonen, A. Trave, R. Car, N. Marzari, A. Kokalj, `http://www.pwscf.org/`.

2 Installation

Presently, the Quantum-ESPRESSO package is only distributed in source form; some precompiled executables (binary files) are provided only for PWgui. Providing binaries would require too much effort and would work only for a small number of machines anyway.

Stable releases of the Quantum-ESPRESSO source package (current version is 3.0) can be downloaded from this URL:

```
http://www.pwscf.org/download.htm
```

Uncompress and unpack the distribution using the command:

```
tar zxvf espresso-3.0.tar.gz
```

If your version of `tar` doesn't recognize the `z` flag, use this instead:

```
gunzip -c espresso-3.0.tar.gz | tar xvf -
```

`cd` to the directory `espresso/` that will be created. The bravest may access the (unstable) development version via anonymous CVS (Concurrent Version System): see the file `README.cvs` contained in the distribution.

To install Quantum-ESPRESSO from source, you need C and Fortran-95 compilers (Fortran-90 is not sufficient, but most "Fortran-90" compilers are actually Fortran-95-compliant). If you don't have a commercial Fortran-95 compiler, you may install the free `g95` compiler (`http://www.g95.org/`): it is still unfinished but already usable. You also need a minimal Unix environment: basically, a command shell (e.g., `bash` or `tcsh`) and the `make` and `awk` utilities. MS-Windows users need to have Cygwin (a UNIX environment which runs under Windows) installed. See `http://www.cygwin.com/`.

Instructions for the impatient:

```
./configure
make all
```

Executable programs (actually, symlinks to them) will be placed in the `bin/` directory.

If you have problems or would like to tweak the default settings, read the detailed instructions below.

2.1 Configure

To configure the Quantum-ESPRESSO source package, run the `configure` script. It will (try to) detect compilers and libraries available on your machine, and set up things accordingly. Presently it is expected to work on

most Linux 32- and 64-bit (Itanium and Opteron) PCs and clusters, IBM SP machines, SGI Origin, some HP-Compaq Alpha machines, Cray X1, Mac OS X, MS-Windows PCs. It may work with some assistance also on other architectures (see below).

For cross-compilation, you have to specify the target machine with the `--host` option (see below). This feature has not been extensively tested, but we had at least one successful report (compilation for NEC SX6 on a PC).

Specifically, `configure` generates the following files:

`make.sys`: compilation rules and flags
`*/make.depend`: dependencies, per source directory
`configure.msg`: a report of the configuration run

`configure.msg` is only used by `configure` to print its final report. It isn't needed for compilation. `make.depend` files are actually generated by invoking the `makedeps.sh` shell script. If you modify the program sources, you might have to rerun it.

You should always be able to compile the Quantum-ESPRESSO suite of programs without having to edit any of the generated files. However you may have to tune `configure` by specifying appropriate environment variables and/or command-line options. Usually the most tricky part is to get external libraries recognized and used: see section 2.1.1, "Libraries", for details and hints.

Environment variables may be set in any of these ways:

```
export VARIABLE=value          # sh, bash, ksh
./configure

setenv VARIABLE value          # csh, tcsh
./configure

./configure VARIABLE=value     # any shell
```

Some environment variables that are relevant to `configure` are:

`ARCH`: label identifying the machine type (see below)
`F90`, `F77`, `CC`: names of Fortran 95, Fortran 77, and C compilers
`MPIF90`, `MPIF77`, `MPICC`: names of parallel compilers
`CPP`: source file preprocessor (defaults to `$CC -E`)
`LD`: linker (defaults to `$MPIF90`)
`CFLAGS`, `FFLAGS`, `F90FLAGS`, `CPPFLAGS`, `LDFLAGS`: compilation flags
`LIBDIRS`: extra directories to search for libraries (see below)

For example, the following command line:

```
./configure MPIF90=mpf90 FFLAGS="-O2 -assume byterecl" \
            CC=gcc CFLAGS=-O3 LDFLAGS=-static
```

instructs `configure` to use `mpf90` as Fortran 95 compiler with flags `-O2 -assume byterecl`, `gcc` as C compiler with flags `-O3`, and to link with flags `-static`. Note that the value of `FFLAGS` must be quoted, because it contains spaces.

If your machine type is unknown to `configure`, you may use the `ARCH` variable to suggest an architecture among supported ones. Try the one that looks more similar to your machine type; you'll probably have to do some additional tweaking. Currently supported architectures are:

`linux64`: Linux 64-bit machines (Itanium, Opteron)
`linux32`: Linux PCs
`aix`: IBM AIX machines
`mips`: SGI MIPS machines
`alpha`: HP-Compaq alpha machines
`sparc`: Sun SPARC machines
`crayx1`: Cray X1 machines
`mac`: Apple PowerPC machines running Mac OS X
`cygwin`: MS-Windows PCs with Cygwin

Finally, `configure` recognizes the following command-line options:

`--disable-parallel`: compile serial code, even if parallel environment is available.
`--disable-shared`: don't use shared libraries: generate static executables.
`--enable-shared`: use shared libraries.
`--host=`*target*: specify target machine for cross-compilation. *Target* must be a string identifying the architecture that you want to compile for; you can obtain it by running `config.guess` on the target machine.

If you want to modify the `configure` script (advanced users only!), read the instructions in `README.configure` first. You'll need GNU Autoconf (`http://www.gnu.org/software/autoconf/`).

2.1.1 Libraries

Quantum-ESPRESSO makes use of the following external libraries:

- BLAS (`http://www.netlib.org/blas/`) and LAPACK (`http://www.netlib.org/lapack/`) for linear algebra
- FFTW (`http://www.fftw.org/`) for Fast Fourier Transforms

A copy of the needed routines is provided with the distribution. However, when available, optimized vendor-specific libraries can be used instead: this often yields huge performance gains.

Quantum-ESPRESSO can use the following architecture-specific replacements for BLAS and LAPACK:

`essl` for IBM machines
`complib.sgimath` for SGI Origin
`SCSL` for SGI Altix
`scilib` for Cray/T3e
`sunperf` for Sun
`MKL` for Intel Linux PCs
`ACML` for AMD Linux PCs
`cxml` for HP-Compaq Alphas.

If none of these is available, we suggest that you use the optimized ATLAS library (`http://math-atlas.sourceforge.net/`). Note that ATLAS is not a complete replacement for LAPACK: it contains all of the BLAS, plus the LU code, plus the full storage Cholesky code. Follow the instructions in the ATLAS distributions to produce a full LAPACK replacement.

Axel Kohlmeyer maintains a set of ATLAS libraries, containing all of LAPACK and no external reference to fortran libraries:
`http://www.theochem.rub.de/~axel.kohlmeyer/cpmd-linux.html#atlas`

Sergei Lisenkov reported success and good performances with optimized BLAS by Kazushige Goto. They can be downloaded freely (but not redistributed!) from: `http://www.cs.utexas.edu/users/flame/goto/`

The FFTW library can also be replaced by vendor-specific FFT libraries, when available, or you can link to a precompiled FFTW library. Please note that you must use FFTW version 2. Support for version 3 is in progress: contact the developers if you want to try.

Finally, Quantum-ESPRESSO can use the MASS vector math library from IBM, if available (only on AIX).

The `configure` script attempts to find optimized libraries, but may fail if they have been installed in non-standard places. You should examine the final value of `BLAS_LIBS`, `LAPACK_LIBS`, `FFT_LIBS`, `MPI_LIBS` (if needed), `MASS_LIBS` (IBM only), either in the output of `configure` or in the generated `make.sys`, to check whether it found all the libraries that you intend to use.

If any libraries weren't found, you can specify a list of directories to search in the environment variable `LIBDIRS`, and rerun `configure`; directories in the list must be separated by spaces. For example:

```
./configure LIBDIRS="/opt/intel/mkl70/lib/32 /usr/lib/math"
```

If this still fails, you may set some or all of the `*_LIBS` variables manually and retry. For example:

```
./configure BLAS_LIBS="-L/usr/lib/math -lf77blas -latlas_sse"
```

Beware that in this case, `configure` will blindly accept the specified value, and won't do any extra search. This is so that if `configure` finds any library that you don't want to use, you can override it.

If you want to use a precompiled FFTW library, the corresponding `fftw.h` include file is also required. That may or may not have been installed on your system together with the library: in particular, most Linux distributions split libraries into "base" and "development" packages, include files normally belonging to the latter. Thus if you can't find `fftw.h` on your machine, chances are you must install the FFTW development package (how exactly it's called depends on your distribution).

If instead the file is there, but `configure` doesn't find it, you may specify its location in the `INCLUDEFFTW` environment variable. For example:

```
./configure INCLUDEFFTW="/usr/lib/fftw-2.1.3/fftw"
```

If everything else fails, you'll have to write the `make.sys` file manually: see section 2.1.2, "Manual configuration".

Please Note: If you change any settings after a previous (successful or failed) compilation, you must run `make clean` before recompiling, unless you know exactly which routines are affected by the changed settings and how to force their recompilation.

2.1.2 Manual configuration

To configure Quantum-ESPRESSO manually, you have to write a working `make.sys` yourself, and run `makedeps.sh` to generate `*/make.depend` files.

For `make.sys`, several templates (each for a different machine type) to start with are provided in the `install/` directory: they have names of the form `Make.`*system*, where *system* is a string identifying the architecture and compiler. Currently available systems are:

`alpha`: HP-Compaq alpha workstations
`alphaMPI`: HP-Compaq alpha parallel machines
`altix`: SGI Altix 350/3000 with Linux, Intel compiler
`beo_ifc`: Linux clusters of PCs, Intel compiler
`beowulf`: Linux clusters of PCs, Portland compiler
`cygwin`: Windows PC, Intel compiler
`fujitsu`: Fujitsu vector machines
`hitachi`: Hitachi SR8000
`hp`: HP PA-RISC workstations
`hpMPI`: HP PA-RISC parallel machines
`ia64`: HP Itanium workstations
`ibm`: IBM RS6000 workstations
`ibmsp`: IBM SP machines
`irix`: SGI workstations
`origin`: SGI Origin 2000/3000
`pc_abs`: Linux PCs, Absoft compiler
`pc_ifc`: Linux PCs, Intel compiler
`pc_lahey`: Linux PCs, Lahey compiler
`pc_pgi`: Linux PCs, Portland compiler
`sun`: Sun workstations
`sunMPI`: Sun parallel machines
`sxcross`: NEC SX-6 (cross-compilation)
`t3e`: Cray T3E

To select the appropriate templates, you can run:

`./configure.old` *system*

where *system* is the best match to your configuration; `configure.old` with no arguments prints the up-to-date list of available systems.

That will copy `Make.`*system* to `make.sys`; for convenience, it'll also run `makedeps.sh` to generate `*/make.depend` files.

Most probably (and even more so if there isn't an exact match to your machine type), you'll have to tweak `make.sys` by hand. In particular, you must specify the full list of libraries that you intend to link to. You'll also have to set the `MYLIB` variable to:

`blas_and_lapack` to compile BLAS and LAPACK from source;
`lapack_mkl` to use the Intel MKL library;
`lapack_t3e` to use the LAPACK for Cray T3E;
otherwise, leave it empty.

Note for HP PA-RISC users: The Makefile for HP PA-RISC workstations and parallel machines is based on a Makefile contributed by Sergei Lysenkov. It assumes that you have HP compiler with MLIB libraries installed on a machine running HP-UX.

Note for MS-Windows users: The Makefile for Windows PCs is based on a Makefile written for an earlier version of PWscf (1.2.0), contributed by Lu Fu-Fa, CCIT, Taiwan. You will need the Cygwin package. The provided Makefile assumes that you have the Intel compiler with MKL libraries installed. It is untested.

If you run into trouble, a possibility is to install Linux in dual-boot mode. You need to create a partition for Linux, install it, install a boot loader (LILO, GRUB). The latter step is not needed if you boot from floppy or CD-ROM. In principle one could avoid installation altogether using a distribution like Knoppix that runs directly from CD-ROM, but for serious use disk access is needed.

2.2 Compile

There are a few adjustable parameters in `Modules/parameters.f90`. The present values will work for most cases. All other variables are dynamically allocated: you do not need to recompile your code for a different system.

At your option, you may compile the complete Quantum-ESPRESSO suite of programs (with `make all`), or only some specific programs.

`make` with no arguments yields a list of valid compilation targets. Here is a list:

- `make pw` produces `PW/pw.x` and `PW/memory.x`.

  `pw.x` calculates electronic structure, structural optimization, molecular dynamics, barriers with NEB. `memory.x` is an auxiliary program that checks the input of `pw.x` for correctness and yields a rough (under-) estimate of the required memory.

- `make ph` produces `PH/ph.x`.

  `ph.x` calculates phonon frequencies and displacement patterns, dielectric tensors, effective charges (uses data produced by `pw.x`).

- `make d3` produces `D3/d3.x`

  `d3.x` calculates anharmonic phonon lifetimes (third-order derivatives of the energy), using data produced by `pw.x` and `ph.x` (Ultrasoft pseudopotentials not supported).

- `make gamma` produces `Gamma/phcg.x`.

  `phcg.x` is a version of `ph.x` that calculates phonons at $\mathbf{q} = 0$ using conjugate-gradient minimization of the density functional expanded to second-order. Only the Γ ($\mathbf{q} = 0$) point is used for Brillouin zone integration. It is faster and takes less memory than `ph.x`, but does not support Ultrasoft pseudopotentials.

- `make pp` produces several codes for data postprocessing, in `PP/` (see list below).

- `make tools` produces several utility programs, mostly for phonon calculations, in `pwtools/` (see list below).

- `make pwcond` produces `PWCOND/pwcond.x`, for ballistic conductance calculations (experimental).

- `make pwall` produces all of the above.

- `make ld1` produces code `atomic/ld1.x` for pseudopotential generationd (see the specific documentation in `atomic_doc/`).

- `make upf` produces utilities for pseudopotential conversion in directory `upftools/` (see section 4, "Pseudopotentials").

- `make cp` produces the Car-Parrinello code CP in `CPV/cp.x`. and the postprocessing code `CPV/cppp.x`.

- `make all` produces all of the above.

For the setup of the GUI, refer to the `PWgui-`*X.Y.Z*`/INSTALL` file, where *X.Y.Z* stands for the version number of the GUI (should be the same as the general version number, currently 3.0). If you are using the CVS-sources, see the `GUI/README` file instead.

The codes for data postprocessing in `PP/` are:

- `pp.x` extracts the specified data from files produced by `pw.x`, prepare data fro plotting by writing them into formats that can be read by several plotting programs

- `bands.x` extracts and reorders eigenvalues from files produced by `pw.x` for band structure plotting

- `projwfc.x` calculates projections of wavefunction over atomic orbitals, performs Löwdin population analysis and calculates projected density of states. These can be summed using auxiliary code `sumpdos.x`

- `dipole.x` calculates the dipole moment for isolated systems (molecules) and the Makov-Payne correction for molecules in supercells (beware: meaningful results only if the charge density is completely contained into the Wigner-Seitz cell)
- `plotrho.x` produces PostScript 2-d contour plots
- `plotband.x` reads the output of `bands.x`, produces band structure PostScript plots
- `average.x` calculates planar averages of potentials
- `voronoy.x` divides the charge density into Voronoy polyhedra (obsolete, use at your own risk)
- `dos.x` calculates electronic Density of States (DOS)
- `pw2wan.x`: interface with code WanT for calculation of transport properties via Wannier (also known as Boyd) functions: see `http://www.wannier-transport.org/`
- `pmw.x` generates Poor Man's Wannier functions, to be used in LDA+U calculations
- `pw2casino.x`: interface with CASINO code for Quantum Monte Carlo calculation (`http://www.tcm.phy.cam.ac.uk/~mdt26/casino.html`).

The utility programs in `pwtools/` are:

- `dynmat.x` applies various kinds of Acoustic Sum Rule (ASR), calculates LO-TO splitting at $\mathbf{q} = 0$ in insulators, IR and Raman cross sections (if the coefficients have been properly calculated), from the dynamical matrix produced by `ph.x`
- `q2r.x` calculates Interatomic Force Constants (IFC) in real space from dynamical matrices produced by `ph.x` on a regular $\mathbf{q}$-grid
- `matdyn.x` produces phonon frequencies at a generic wave vector using the IFC file calculated by `q2r.x`; may also calculate phonon DOS
- `fqha.x` for quasi-harmonic calculations
- `lambda.x` calculates the electron-phonon coefficient λ and the function $\alpha^2 F(\omega)$

- `dist.x` calculates distances and angles between atoms in a cell, taking into account periodicity
- `ev.x` fits energy-vs-volume data to an equation of state
- `kpoints.x` produces lists of k-points
- `pwi2xsf.sh`, `pwo2xsf.sh` process respectively input and output files (not data files!) for `pw.x` and produce an XSF-formatted file suitable for plotting with XCrySDen, a powerful crystalline and molecular structure visualization program (`http://www.xcrysden.org/`). BEWARE: the `pwi2xsf.sh` shell script requires the `pwi2xsf.x` executables to be located somewhere in your `$PATH`.
- `band_plot.x`: undocumented and possibly obsolete
- `bs.awk`, `mv.awk` are scripts that process the output of `pw.x` (not data files!). Usage:

  ```
  awk -f bs.awk < my-pw-file > myfile.bs
  awk -f mv.awk < my-pw-file > myfile.mv
  ```

  The files so produced are suitable for use with `xbs`, a very simple X-windows utility to display molecules, available at:
  `http://www.ccl.net/cca/software/X-WINDOW/xbsa/README.shtml`
- `path_int.sh/path_int.x`: utility to generate, starting from a path (a set of images), a new one with a different number of images. The initial and final points of the new path can differ from those in the original one. Useful for NEB calculations.
- `kvecs_FS.x, bands_FS.x`: utilities for Fermi Surface plotting usingX CrySDen

2.3 Run examples

As a final check that compilation was successful, you may want to run some or all of the examples contained within the `examples` directory of the Quantum-ESPRESSO distribution. Those examples try to exercise all the programs and features of the Quantum-ESPRESSO package. A list of examples and of what each example does is contained in `examples/README`. For details, see the `README` file in each example's directory. If you find that any relevant

feature isn't being tested, please contact us (or even better, write and send us a new example yourself!).

If you haven't downloaded the full Quantum-ESPRESSO distribution and don't have the examples, you can get them from the Test and Examples Page of the Quantum-ESPRESSO web site (`http://www.pwscf.org/tests.htm`). The necessary pseudopotentials are included.

To run the examples, you should follow this procedure:

1. Go to the `examples` directory and edit the `environment_variables` file, setting the following variables as needed:

   `BIN_DIR=` directory where Quantum-ESPRESSO executables reside
   `PSEUDO_DIR=` directory where pseudopotential files reside
   `TMP_DIR=` directory to be used as temporary storage area

   If you have downloaded the full Quantum-ESPRESSO distribution, you may set `BIN_DIR=$TOPDIR/bin` and `PSEUDO_DIR=$TOPDIR/pseudo`, where `$TOPDIR` is the root of the Quantum-ESPRESSO source tree.

   In order to be able to run all the examples, the `PSEUDO_DIR` directory must contain the following files:

   `Al.vbc.UPF`, `As.gon.UPF`, `C.pz-rrkjus.UPF`, `Cu.pz-d-rrkjus.UPF`, `Fe.pz-nd-rrkjus.UPF`, `H.fpmd.UPF`, `H.vbc.UPF`, `N.BLYP.UPF`, `Ni.pbe-nd-rrkjus.UPF`, `NiUS.RRKJ3.UPF`, `O.BLYP.UPF`, `O.LDA.US.RRKJ3.UPF`, `O.pbe-rrkjus.UPF`, `O.vdb.UPF`, `OPBE_nc.UPF`, `Pb.vdb.UPF`, `Ptrel.RRKJ3.UPF`, `Si.vbc.UPF`, `SiPBE_nc.UPF`, `Ti.vdb.UPF`

   If any of these are missing, you can download them (and many others) from the Pseudopotentials Page of the Quantum-ESPRESSO web site (`http://www.pwscf.org/pseudo.htm`).

   `TMP_DIR` must be a directory you have read and write access to, with enough available space to host the temporary files produced by the example runs, and possibly offering high I/O performance (i.e., don't use an NFS-mounted directory).

2. If you have compiled the parallel version of Quantum-ESPRESSO (this is the default if parallel libraries are detected), you will usually have to specify a driver program (such as `poe` or `mpiexec`) and the number of processors: read section 3, "Running on parallel machines" for details.

In order to do that, edit again the `environment_variables` file and set the `PARA_PREFIX` and `PARA_POSTFIX` variables as needed. Parallel executables will be run by a command like this:

```
$PARA_PREFIX pw.x $PARA_POSTFIX < file.in > file.out
```

For example, if the command line is like this (as for an IBM SP4):

```
poe pw.x -procs 4 < file.in > file.out
```

you should set `PARA_PREFIX="poe"`, `PARA_POSTFIX="-procs 4"`.

Furthermore, if your machine does not support interactive use, you must run the commands specified below through the batch queueing system installed on that machine. Ask your system administrator for instructions.

3. To run a single example, go to the corresponding directory (for instance, `example/example01`) and execute:

   ```
   ./run_example
   ```

   This will create a subdirectory `results`, containing the input and output files generated by the calculation.

   Some examples take only a few seconds to run, while others may require several minutes depending on your system.

   To run all the examples in one go, execute:

   ```
   ./run_all_examples
   ```

   from the `examples` directory. On a single-processor machine, this typically takes one to three hours.

   The `make_clean` script cleans the examples tree, by removing all the `results` subdirectories. However, if additional subdirectories have been created, they aren't deleted.

4. In each example's directory, the `reference` subdirectory contains verified output files, that you can check your results against. They were generated on a Linux PC using the Intel compiler. On different architectures the precise numbers could be slightly different, in particular if different FFT dimensions are automatically selected. For this reason,

a plain `diff` of your results against the reference data doesn't work, or at least, it requires human inspection of the results.

Instead, you can run the `check_example` script in the `examples` directory:

`./check_example` *example_dir*

where *example_dir* is the directory of the example that you want to check (e.g., `./check_example example01`). You can specify multiple directories.

Note: at the moment `check_example` is in early development and (should be) guaranteed to work only on examples 01 to 04.

2.4 Installation issues

The main development platforms are IBM SP and Intel/AMD PC with Linux and Intel compiler. For other machines, we rely on user's feedback.

All machines Working fortran-95 and C compilers are needed in order to compile Quantum-ESPRESSO. Most so-called "fortran-90" compilers implement the fortran-95 standard, but older versions may not be fortran-95 compliant.

If you get "Compiler Internal Error" or similar messages, try to lower the optimization level, or to remove optimization, just for the routine that has problems. If it doesn't work, or if you experience weird problems, try to install patches for your version of the compiler (most vendors release at least a few patches for free), or to upgrade to a more recent version.

If you get an error in the loading phase that looks like "ld: file XYZ.o: unknown (unrecognized, invalid, wrong, missing, ...) file type", or "While processing relocatable file XYZ.o, no relocatable objects were found" (T3E), one of the following things have happened:

1. you have leftover object files from a compilation with another compiler: run `make clean` and recompile.
2. `make` does not stop at the first compilation error (it happens with some compilers). Remove file XYZ.o and look for the compilation error.

If many symbols are missing in the loading phase, you did not specify the location of all needed libraries (LAPACK, BLAS, FFTW, machine-specific optimized libraries). If you did, but symbols are still missing, see below (for Linux PC).

SGI machines with MIPS compiler Many versions of the MIPS compiler yield compilation errors in conjunction with with `FORALL` constructs. There is no known solution other than editing the `FORALL` construct that gives a problem, or to replace it with an equivalent `DO...END DO` construct.

Linux Alphas with Compaq compiler If at linking stage you get error messages like: "undefined reference to 'for_check_mult_overflow64' " with Compaq/HP fortran compiler on Linux Alphas, check the following page: `http://linux.iol.unh.edu/linux/fortran/faq/cfal-X1.0.2.html`.

Linux PC The web site of Axel Kohlmeyer contains a very informative section on compiling and running CPMD on Linux. Most of its contents applies to the Quantum-ESPRESSO code as well:
`http://www.theochem.rub.de/~axel.kohlmeyer/cpmd-linux.html`.

On newer Linux machines, even statically linked binaries will try to open some shared libraries, which will lead to crashes if libc/libm/libpthreads are not linked dynamically. Machines using glibc-2.2.4 and older seem ok: compile on these machines if you want to share precompiled binaries. Crashes due to multithreading (e.g. when using a multithreaded ATLAS or MKL) on machines with the newer threads (nptl) can be worked around by setting the environment variable `LD_ASSUME_KERNEL` to '2.2.5'. For the newest Intel compilers, `-static-libcxa` does the trick most of the time. (info from Axel Kohlmeyer)

Since there is no standard compiler for Linux, different compilers have different ideas about the right way to call external libraries. As a consequence you may have a mismatch between what your compiler calls ("symbols") and the actual name of the required library call. Use the `nm` command to determine the name of a library call, as in the following examples:

```
nm /usr/local/lib/libblas.a | grep T | grep -i daxpy
nm /usr/local/lib/liblapack.a | grep T | grep -i zhegv
```

where typical location and name of libraries is assumed. Most precompiled libraries have lowercase names with one or two underscores (_) appended. `configure` should select the appropriate preprocessing options in `make.sys`, but in case of trouble, be aware that:

- the Absoft compiler is case-sensitive (like C and unlike other Fortran compilers) and does not add an underscore to symbol names (note that if your libraries contain uppercase or mixed case names, you are out of luck: You must either recompile your own libraries, or change the `#define`'s in `include/f_defs.h`);

- both Portland compiler (pgf90) and Intel compiler (ifort/ifc) are case insensitive and add an underscore to symbol names.

With some precompiled lapack libraries, you may need to add `-lg2c` or `-lm` or both.

Linux PCs with Portland Group compiler (pgf90)
Quantum-ESPRESSO does not work reliably, or not at all, with some versions (in particular, 5.2) of the Portland Group compiler. We think that this is due to compiler bugs, not to Quantum-ESPRESSO bugs. In any event, use the latest version of each release of the compiler, with patches if available: see the Portland Group web site,
`http://www.pgroup.com/faq/install.htm#release_info`

Linux PCs (Pentium) with Intel compiler (ifort, formerly ifc)
If `configure` doesn't find the compiler, or if you get "Error loading shared libraries..." at run time, you have forgotten to execute the script that sets up the correct path and library path. Unless your system manager has done this for you, you should execute the appropriate script — located in the directory containing the compiler executable — in your initialization files. Consult the documentation provided by Intel.

Each major release of the Intel compiler differs a lot from the previous one. Do not mix compiled objects from different releases: they are incompatible. Intel compiler v. 7 and later use a different method to locate where modules are with respect to v. < 7: if you are using the manual configuration, choose the appropriate line `MODULEFLAG=...` in `make.sys`.

Some releases of Intel compiler v. 7 and 8 yield "Compiler Internal Error". Update to the last version (presently 7.1.41, 8.0.046 or 8.1.018, respectively), available via Intel Premier support (registration free of charge for Linux): `http://developer.intel.com/software/products/support/#premier`.
There are conflicting reports on the newest version 9. In any event, look for the last version with the most patches.

Warnings "size of symbol ... changed ..." are produced by ifc 7.1 at the loading stage. These seem to be harmless, but they may cause the loader to stop, depending on your system configuration. If this happens and no executable is produced, add the following to `LDFLAGS`: `-Xlinker --noinhibit-exec`.

On Intel CPUs, it is very convenient to use Intel MKL libraries. If `configure` doesn't find them, try `configure --enable-shared`. MKL also contains optimized FFT routines, but they are presently not supported: use FFTW instead. Note that Intel compiler v. 8 fails to load with MKL v. 5.2

or earlier versions, because some symbols that are referenced by MKL are missing. There is a fix for this (info from Konstantin Kudin): add libF90.a from ifc 7.1 at the linking stage, as the last library. Note that some combinations of not-so-recent versions of MKL and ifc may yield a lot of "undefined references" when statically loaded: use `configure --enable-shared`, or remove the `-static` option in `make.sys`. Note that `pwcond.x` works only with recent versions (v.7 or later) of MKL.

When using/testing/benchmarking MKL on SMP (multiprocessor) machines, one should set the environmental variable `OMP_NUM_THREADS` to 1, unless the OpenMP parallelization is desired. MKL by default sets the variable to the number of CPUs installed and thus gives the impression of a much better performance, as the CPUu time is only measured for the master thread (info from Axel Kohlmeyer).

The I/O libraries used by older versions of the Intel compiler are incompatible with those called by most precompiled BLAS/LAPACK libraries (including ATLAS): you get error messages at linking stage. A workaround is to recompile BLAS/LAPACK with ifc, or (better) to replace the BLAS routine `xerbla` and LAPACK routine `dlamch` (the only two containing I/O calls) with recompiled objects:

```
ifc -c xerbla.f
ifc -O0 -c dlamch.f
```

(do not forget `-O0` — `dlamch.f` *must* be compiled without optimization) and replace them into the library, as in the following example:

```
ar rv libatlas.a xerbla.o dlamch.o
```

(assuming that the library and the two object files are in the same directory). See also Axel Kohlmeyer's web site.

Linux distributions using glibc 2.3 or later (such as e.g. RedHat 9) may be incompatible with ifc 7.0 and 7.1. The incompatibility shows up in the form of messages "undefined reference to 'errno' " at linking stage. A workaround is available: see `http://newweb.ices.utexas.edu/misc/ctype.c`.

There is a well known problem with version 8 of Intel compiler and pthreads (that are used both in Debian Woody and Sarge) that causes "segmentation fault" errors (info from Lucas Fernandez Seivane). Version 7 does not have this problem.

AMD CPUs, Intel Itanium AMD Athlon CPUs can be basically treated like Intel Pentium CPUs. You can use the Intel compiler and MKL with Pentium-3 optimization.

Konstantin Kudin reports that the best results in terms of performances are obtained with ATLAS optimized BLAS/LAPACK libraries, using AMD Core Math Library (ACML) for the missing libraries. ACML can be freely downloaded from AMD web site. Beware: some versions of ACML – i.e. the GCC version with SSE2 – crash PWscf. The "_nosse2" version appears to be stable. Load first ATLAS, then ACML, then `-lg2c`, as in the following example (replace what follows `-L` with something appropriate to your configuration):

```
-L/location/of/fftw/lib/ -lfftw \
-L/location/of/atlas/lib -lf77blas -llapack -lcblas -latlas \
-L/location/of/gnu32_nosse2/lib -lacml -lg2c
```

64-bit CPUs like the AMD Opteron and the Intel Itanium are supported and should work both in 32-bit emulation and in 64-bit mode (in the latter case, `-D__LINUX64` is needed among the preprocessing flags). Both the PGI and the Intel compiler (v8.1 EM64T-edition, available via Intel Premier support) should work. 64-bit executables can address a much larger memory space, but apparently they are not especially faster than 32-bit executables. The Intel compiler has been reported to be more reliable and to produce faster executables wrt the PGI compiler. You may also try with g95.

Linux PC clusters with MPI PC clusters running some version of MPI are a very popular computational platform nowadays. Two major MPI implementations (MPICH, LAM-MPI) are available. The number of possible configurations, in terms of type and version of the MPI libraries, kernels, system libraries, compilers, is very large. Quantum-ESPRESSO compiles and works on all non-buggy, properly configured configuration. You may have to recompile MPI libraries in order to be able to use them with the Intel compiler. See Axel Kohlmeyer's web site for precompiled versions of the MPI libraries.

If Quantum-ESPRESSO does not work for some reason on a PC cluster, try first if it works in serial execution. A frequent problem with parallel execution is that Quantum-ESPRESSO does not read from standard input, due to a bad configuration of MPI libraries: see section "Running on parallel machines". If you get weird errors with LAM-MPI, add `-D__LAM` to preprocessing options and recompile. See also Axel Kohlmeyer's web site for more info.

If you are dis satisfied with the performances in parallel execution, read the "Parallelization issues" section.

T3E The following workaround is needed: in files `PW/bp_zgefa.f` and `PW/bp_zgedi.f`, replace all occurrences of `zscal`, `zaxpy`, `zswap`, `izamax` with `cscal`, `caxpy`, `cswap`, `icamax`. Also, in `PP/dist.f` you need to comment the call to `getarg` and uncomment the call to `pxfgetarg`.

If you have a T3E with "benchlib" installed, you may want to use it by adding `-D__BENCHLIB` to preprocessing flags. If you get errors at loading because symbols `LPUTP`, `LGETV`, `LSETV` are undefined, you either need to link "benchlib", or to remove `-D__BENCHLIB` and recompile (after a `make clean`).

3 Running on parallel machines

Parallel execution is strongly system- and installation-dependent. Typically one has to specify:

- a launcher program, such as `poe`, `mpirun`, or `mpiexec`;
- the number of processors, typically as an option to the launcher program, but in some cases *after* the program to be executed;
- the program to be executed, with the proper path if needed: for instance, `pw.x`, or `./pw.x`, or `$HOME/bin/pw.x`, or whatever applies;
- the number of "pools" into which processors are to be grouped (see section 7.4, "Parallelization Issues", for an explanation of what a pool is).

The last item is optional and is read by the code. The first and second items are machine- and installation-dependent, and may be different for interactive and batch execution.

Please note: Your machine might be configured so as to disallow interactive execution: if in doubt, ask your system administrator.

For illustration, here's how to run `pw.x` on 16 processors partitioned into 8 pools (2 processors each), for several typical cases. For convenience, we also give the corresponding values of `PARA_PREFIX`, `PARA_POSTFIX` to be used in running the examples distributed with Quantum-ESPRESSO (see section 2.3, "Run examples").

IBM SP machines, batch:

```
pw.x -npool 8 < input

PARA_PREFIX="", PARA_POSTFIX="-npool 8"
```

This should also work interactively, with environment variables `NPROC` set to 16, `MP_HOSTFILE` set to the file containing a list of processors.

IBM SP machines, interactive, using `poe`:

```
poe pw.x -procs 16 -npool 8 < input

PARA_PREFIX="poe", PARA_POSTFIX="-procs 16 -npool 8"
```

SGI Origin and PC clusters using `mpirun`:

```
mpirun -np 16 pw.x -npool 8 < input
```

```
PARA_PREFIX="mpirun -np 16", PARA_POSTFIX="-npool 8"
```

PC clusters using `mpiexec`:

```
mpiexec -n 16 pw.x -npool 8 < input
```

```
PARA_PREFIX="mpiexec -n 16", PARA_POSTFIX="-npool 8"
```

Cray T3E (old):

```
mpprun -n 16 pw.x -npool 8 < input
```

```
PARA_PREFIX="mpprun -n 16", PARA_POSTFIX="-npool 8"
```

Note that each processor writes its own set of temporary wavefunction files during the calculation. If `wf_collect=.true.` (in namelist `control`), the final result is collected into a single file, whose format is independent on the number of processors; otherwise, one wavefunction file per processor is left on the disk. In the latter case, the files are readable only by a job running on the same number of processors and pools, and if all files are on a file system that is visible to all processors (i.e., you cannot use local scratch directories: there is presently no way to ensure that the distribution of processes on processors will follow the same pattern for different jobs).

Some implementations of the MPI library may have problems with input redirection in parallel. If this happens, use the option `-in` (or `-inp` or `-input`), followed by the input file name. Example: `pw.x -in input -npool 4 > output`.

Please note that all postprocessing codes *not* reading data files produced by `pw.x` — that is, `average.x`, `voronoy.x`, `dos.x` — the plotting codes `plotrho.x`, `plotband.x`, and all executables in `pwtools/`, should be executed on just one processor. Unpredictable results may follow if those codes are run on more than one processor.

4 Pseudopotentials

Currently PWscf and CP support both Ultrasoft (US) Vanderbilt pseudopotentials (PPs) and Norm-Conserving (NC) Hamann-Schlüter-Chiang PPs in separable Kleinman-Bylander form. Note however that calculation of third-order derivatives is not (yet) implemented with US PPs.

The Quantum-ESPRESSO package uses a unified pseudopotential format (UPF) (`http://www.pwscf.org/format.htm`) for all types of PPs, but still accepts a number of other formats:

- the "old PWscf" format for NC PPs (PWscf only!),
- the "old CP" format for NC PPs (CP only!),
- the "old FPMD" format for NC PPs (CP only!),
- the "new PWscf" format for NC and US PPs,
- the "Vanderbilt" format (formatted, not binary) for NC and US PPs.

See also `http://www.pwscf.org/oldformat.htm`.

A large collection of PPs (currently about 60 elements covered) can be downloaded from the Pseudopotentials Page of the Quantum-ESPRESSO web site (`http://www.pwscf.org/pseudo.htm`). The naming convention for these PPs is explained in file `Doc/nomefile.upf`.

If you do not find there the PP you need (because there is no PP for the atom you need or you need a different exchange-correlation functional or a different core-valence partition or for whatever reason may apply), it may be taken, if available, from published tables, such as e.g.:

- G.B. Bachelet, D.R. Hamann and M. Schlüter, Phys. Rev. B **26**, 4199 (1982)
- X. Gonze, R. Stumpf, and M. Scheffler, Phys. Rev. B **44**, 8503 (1991)
- S. Goedecker, M. Teter, and J. Hutter, Phys. Rev. B **54**, 1703 (1996)

or otherwise it must be generated. Since version 2.1, Quantum-ESPRESSO includes a PP generation package, in the directory `atomic/` (sources) and `atomic_doc/` (documentation, tests and examples). The package can generate both NC and US PPs in UPF format. We refer to its documentation for instructions on how to generate PPs with the `atomic/` code.

Other PP generation packages are available on-line:

- David Vanderbilt's code (UltraSoft PPs):
  `http://www.physics.rutgers.edu/~dhv/uspp/index.html`
- Fritz Haber's code (Norm-Conserving PPs):
  `http://www.fhi-berlin.mpg.de/th/fhi98md/fhi98PP`
- José-Luís Martins' code (Norm-Conserving PPs):
  `http://bohr.inesc-mn.pt/~jlm/pseudo.html`

The first two codes produce PPs in UPF format, or in a format that can be converted to unified format using the utilities of directory `upftools/`.

Finally, other electronic-structure packages (CAMPOS, ABINIT) provide tables of PPs that can be freely downloaded, but need to be converted into a suitable format for use with Quantum-ESPRESSO.

Remember: *always* test the PPs on simple test systems before proceeding to serious calculations.

5 Using PWscf

Input files for the PWscf codes may be either written by hand (the good old way), or produced via the "PWgui" graphical interface by Anton Kokalj, included in the Quantum-ESPRESSO distribution. See `PWgui-`*x.y.z*`/INSTALL` (where *x.y.z* is the version number) for more info on PWgui, or `GUI/README` if you are using CVS sources.

You may take the examples distributed with Quantum-ESPRESSO as templates for writing your own input files: see section 2.3, "Run examples". In the following, whenever we mention "Example N", we refer to those. Input files are those in the `results` directories, with names ending in `.in` (they'll appear after you've run the examples).

Note about exchange-correlation: the type of exchange-correlation used in the calculation is read from PP files. All PP's must have been generated using the same exchange-correlation.

5.1 Electronic and ionic structure calculations

Electronic and ionic structure calculations are performed by program `pw.x`.

5.1.1 Input data

The input data is organized as several namelists, followed by other fields introduced by keywords.

The namelists are

`&CONTROL`: general variables controlling the run
`&SYSTEM`: structural information on the system under investigation
`&ELECTRONS`: electronic variables: self-consistency, smearing
`&IONS` (optional): ionic variables: relaxation, dynamics
`&CELL` (optional): variable-cell dynamics
`&PHONON` (optional): information required to produce data for phonon calculations

Optional namelist may be omitted if the calculation to be performed does not require them. This depends on the value of variable `calculation` in namelist `&CONTROL`. Most variables in namelists have default values. Only the following variables in `&SYSTEM` must always be specified:

`ibrav` (integer): bravais-lattice index
`celldm` (real, dimension 6): crystallographic constants

`nat` (integer): number of atoms in the unit cell
`ntyp` (integer): number of types of atoms in the unit cell
`ecutwfc` (real): kinetic energy cutoff (Ry) for wavefunctions.

For metallic systems, you have to specify how metallicity is treated by setting variable `occupations`. If you choose `occupations='smearing'`, you have to specify the smearing width `degauss` and optionally the smearing type `smearing`. If you choose `occupations='tetrahedra'`, you need to specify a suitable uniform k-point grid (card `K_POINTS` with option `automatic`). Spin-polarized systems must be treated as metallic system, except the special case of a single k-point, for which occupation numbers can be fixed (`occupations='from_input'` and card `OCCUPATIONS`).

Explanations for the meaning of variables `ibrav` and `celldm` are in file `INPUT_PW`. Please read them carefully. There is a large number of other variables, having default values, which may or may not fit your needs.

After the namelists, you have several fields introduced by keywords with self-explanatory names:

`ATOMIC_SPECIES`
`ATOMIC_POSITIONS`
`K_POINTS`
`CELL_PARAMETERS` (optional)
`OCCUPATIONS` (optional)
`CLIMBING_IMAGES` (optional)

The keywords may be followed on the same line by an option. Unknown fields (including some that are specific to CP code) are ignored by PWscf. See file `Doc/INPUT_PW` for a detailed explanation of the meaning and format of the various fields.

Note about k points: The k-point grid can be either automatically generated or manually provided as a list of k-points and a weight in the Irreducible Brillouin Zone only of the *Bravais lattice* of the crystal. The code will generate (unless instructed not to do so: see variable `nosym`) all required k-points and weights if the symmetry of the system is lower than the symmetry of the Bravais lattice. The automatic generation of k-points follows the convention of Monkhorst and Pack.

5.1.2 Typical cases

We may distinguish the following typical cases for `pw.x`:

single-point (fixed-ion) SCF calculation. Set `calculation='scf'`.

Namelists &IONS and &CELL need not to be present (this is the default). See Example 01.

band structure calculation. First perform a SCF calculation as above; then do a non-SCF calculation specifying calculation='nscf', with the desired k-point grid and number nbnd of bands.

Specify nosym=.true. to avoid generation of additional k-points in low symmetry cases. Variables prefix and outdir, which determine the names of input or output files, should be the same in the two runs. See Example 01.

structural optimization. Specify calculation='relax' and add namelist &IONS.

All options for a single SCF calculation apply, plus a few others. You may follow a structural optimization with a non-SCF band-structure calculation, but do not forget to update the input ionic coordinates. See Example 03.

molecular dynamics. Specify calculation='md' and time step dt.

Use variable ion_dynamics in namelist &IONS for a fine-grained control of the kind of dynamics. Other options for setting the initial temperature and for thermalization using velocity rescaling are available. Remember: this is MD on the electronic ground state, not Car-Parrinello MD. See Example 04.

polarization via Berry Phase. See Example 10, its README, and the documentation in the header of PW/bp_c_phase.f90.

Nudged Elastic Band calculation. Specify calculation='neb' and add namelist &IONS.

All options for a single SCF calculation apply, plus a few others. In the namelist &IONS the number of images used to discretize the elastic band must be specified. All other variables have a default value. Coordinates of the initial and final image of the elastic band have to be specified in the ATOMIC_POSITIONS card. A detailed description of all input variables is contained in the file Doc/INPUT_PW. See also Example 17.

The output data files are written in the directory specified by variable outdir, with names specified by variable prefix (a string that is prepended to all file names, whose default value is: prefix='pwscf').

The execution stops if you create a file `prefix.EXIT` in the working directory. Note that just killing the process may leave the output files in an unusable state.

5.2 Phonon calculations

The phonon code `ph.x` calculates normal modes at a given **q**-vector, starting from data files produced by `pw.x`.

If $\mathbf{q} = 0$, the data files can be produced directly by a simple SCF calculation. For phonons at a generic **q**-vector, you need to perform first a SCF calculation, then a band-structure calculation (see above) with `calculation = 'phonon'`, specifying the **q**-vector in variable `xq` of namelist `&PHONON`.

The output data file appear in the directory specified by variables `outdir`, with names specified by variable `prefix`. After the output file(s) has been produced (do not remove any of the files, unless you know which are used and which are not), you can run `ph.x`.

The first input line of `ph.x` is a job identifier. At the second line the namelist `&INPUTPH` starts. The meaning of the variables in the namelist (most of them having a default value) is described in file `INPUT_PH`. Variables `outdir` and `prefix` must be the same as in the input data of `pw.x`. Presently you must also specify `amass` (real, dimension `ntyp`): the atomic mass of each atomic type.

After the namelist you must specify the **q**-vector of the phonon mode. This must be the same **q**-vector given in the input of `pw.x`.

Notice that the dynamical matrix calculated by `ph.x` at $\mathbf{q} = 0$ does not contain the non-analytic term occuring in polar materials, i.e. there is no LO-TO splitting in insulators. Moreover no Acoustic Sum Rule (ASR) is applied. In order to have the complete dynamical matrix at $\mathbf{q} = 0$ including the non-analytic terms, you need to calculate effective charges by specifying option `epsil=.true.` to `ph.x`.

Use program `dynmat.x` to calculate the correct LO-TO splitting, IR cross sections, and to impose various forms of ASR. If `ph.x` was instructed to calculate Raman coefficients, `dynmat.x` will also calculate Raman cross sections for a typical experimental setup.

A sample phonon calculation is performed in Example 02.

5.2.1 Calculation of interatomic force constants in real space

First, dynamical matrices $D(\mathbf{q})$ are calculated and saved for a suitable uniform grid of **q**-vectors (only those in the Irreducible Brillouin Zone of the crystal are needed). Although this can be done one **q**-vector at the time,

a simpler procedure is to specify variable `ldisp=.true` and to set variables `nq1,nq2,nq3` to some suitable Monkhorst-Pack grid, that will be automatically generated, centered at $\mathbf{q} = 0$. Do not forget to specify `epsil=.true.` in the input data of `ph.x` if you want the correct TO-LO splitting in polar materials.

Second, code `q2r.x` reads the $D(\mathbf{q})$ dynamical matrices produced in the preceding step and Fourier-transform them, writing a file of Interatomic Force Constants in real space, up to a distance that depends on the size of the grid of $\mathbf{q}$-vectors. Program `matdyn.x` may be used to produce phonon modes and frequencies at any $\mathbf{q}$ using the Interatomic Force Constants file as input.

See Example 06.

5.2.2 Calculation of electron-phonon interaction coefficients

The calculation of electron-phonon coefficients in metals is made difficult by the slow convergence of the sum at the Fermi energy. It is convenient to calculate phonons, for each $\mathbf{q}$-vector of a suitable grid, using a smaller k-point grid, saving the dynamical matrix and the self-consistent first-order variation of the potential (variable `fildvscf`). Then a non-SCF calculation with a larger k-point grid is performed. Finally the electron-phonon calculation is performed by specifying `elph=.true.`, `trans=.false.`, and the input files `fildvscf`, `fildyn`. The electron-phonon coefficients are calculated using several values of gaussian broadening (see `PH/elphon.f90`) because this quickly shows whether results are converged or not with respect to the k-point grid and Gaussian broadening. See Example 07.

All of the above must be repeated for all desired $\mathbf{q}$-vectors and the final result is summed over all $\mathbf{q}$-vectors, using `pwtools/lambda.x`. The input data for the latter is described in the header of `pwtools/lambda.f90`.

5.3 Post-processing

There are a number of auxiliary codes performing postprocessing tasks such as plotting, averaging, and so on, on the various quantities calculated by `pw.x`. Such quantities are saved by `pw.x` into the output data file(s).

The main postprocessing code `pp.x` reads data file(s), extracts or calculated the selected quantity, writes it into a format that is suitable for plotting. Quantities that can be read or calculated are:

charge density
spin polarization
various potentials

local density of states at E_F
local density of electronic entropy
STM images
wavefunction squared
electron localization function
planar averages
integrated local density of states

Various types of plotting (along a line, on a plane, three-dimensional, polar) and output formats (including the popular `cube` format) can be specified. The output files can be directly read by the free plotting system Gnuplot (1D or 2D plots), or by code `plotrho.x` that comes with PWscf (2D plots), or by advanced plotting software XCrySDen and gOpenMol (3D plots)

See file `INPUT_PP` for a detailed description of the input for code `pp.x`. See Example 05 for a charge density plot.

The postprocessing code `bands.x` reads data file(s), extracts eigenvalues, regroups them into bands (the algorithm used to order bands and to resolve crossings may not work in all circumstances, though). The output is written to a file in a simple format that can be directly read by plotting program `plotband.x`. Unpredictable plots may results if **k**-points are not in sequence along lines. See Example 05 for a simple band plot.

The postprocessing code `projwfc.x` calculates projections of wavefunction over atomic orbitals. The atomic wavefunctions are those contained in the pseudopotential file(s). The Löwdin population analysis (similar to Mulliken analysis) is presently implemented. The projected DOS (PDOS, the DOS projected onto atomic orbitals) can also be calculated and written to file(s). More details on the input data are found in the header of file `PP/projwfc.f90`. The auxiliary code `sumpdos.x` (courtesy of Andrea Ferretti) can be used to sum selected PDOS, by specifiying the names of files containing the desired PDOS. Type `sumpdos.x -h` or look into the source code for more details. The total electronic DOS is instead calculated by code `PP/dos.x`. See Example 08 for total and projected electronic DOS calculations.

The postprocessing code `path_int.x` is intended to be used in the framework of NEB calculations. It is a tool to generate a new path (what is actually generated is the restart file) starting from an old one through interpolation (cubic splines). The new path can be discretized with a different number of images (this is its main purpose), images are equispaced and the interpolation can be also performed on a subsection of the old path. The input file needed by `path_int.x` can be easily set up with the help of the self explanatory `path_int.sh` shell script.

6 Using CP

This section is intended to explain how to perform basic Car-Parrinello (CP) simulations using the CP codes.

It is important to understand that a CP simulation is a sequence of different runs, some of them used to "prepare" the initial state of the system, and other performed to collect statistics, or to modify the state of the system itself, i.e. modify the temperature or the pressure.

To prepare and run a CP simulation you should:

1. define the system:
   (a) atomic positions
   (b) system cell
   (c) pseudopotentials
   (d) number of electrons and bands
   (e) cut-offs
   (f) FFT grids (CP code only)
2. The first run, when starting from scratch, is always an electronic minimization, with fixed ions and cell, to bring the electronic system on the ground state (GS) relative to the starting atomic configuration. Example of input file (Benzene Molecule):

```
&control
   title = ' Benzene Molecule ',
   calculation = 'cp',
   restart_mode = 'from_scratch',
   ndr = 51,
   ndw = 51,
   nstep  = 100,
   iprint = 10,
   isave  = 100,
   tstress = .TRUE.,
   tprnfor = .TRUE.,
   dt    = 5.0d0,
   etot_conv_thr = 1.d-9,
   ekin_conv_thr = 1.d-4,
   prefix = 'c6h6'
   pseudo_dir='/scratch/acv0/benzene/',
```

```
   outdir='/scratch/acv0/benzene/Out/'
 /
 &system
   ibrav = 14,
   celldm(1) = 16.0,
   celldm(2) = 1.0,
   celldm(3) = 0.5,
   celldm(4) = 0.0,
   celldm(5) = 0.0,
   celldm(6) = 0.0,
   nat  = 12,
   ntyp = 2,
   nbnd = 15,
   nelec = 30,
   ecutwfc = 40.0,
   nr1b= 10, nr2b = 10, nr3b = 10,
   xc_type = 'BLYP'
 /
 &electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'sd',
 /
 &ions
   ion_dynamics = 'none',
 /
 &cell
   cell_dynamics = 'none',
   press = 0.0d0,
 /
ATOMIC_SPECIES
 C 12.0d0 c_blyp_gia.pp
 H 1.00d0 h.ps
ATOMIC_POSITIONS (bohr)
  C     2.6  0.0 0.0
  C     1.3 -1.3 0.0
  C    -1.3 -1.3 0.0
  C    -2.6  0.0 0.0
  C    -1.3  1.3 0.0
  C     1.3  1.3 0.0
  H     4.4  0.0 0.0
```

```
H     2.2 -2.2 0.0
H    -2.2 -2.2 0.0
H    -4.4  0.0 0.0
H    -2.2  2.2 0.0
H     2.2  2.2 0.0
```

You can find the description of the input variables in file `INPUT_CP` in the `Doc/` directory. A short description of the logic behind the choice of parameters in contained in `INPUT.HOWTO`

3. Sometimes a single run is not enough to reach the GS. In this case, you need to re-run the electronic minimization stage. Use the input of the first run, changing `restart_mode = 'from_scratch'` to `restart_mode = 'restart'`.

   Important: unless you are already experienced with the system you are studying or with the code internals, usually you need to tune some input parameters, like `emass`, `dt`, and cut-offs. For this purpose, a few trial runs could be useful: you can perform short minimizations (say, 10 steps) changing and adjusting these parameters to your need.

   You could specify the degree of convergence with these two thresholds:

   `etot_conv_thr`: total energy difference between two consecutive steps

   `ekin_conv_thr`: value of the fictitious kinetic energy of the electrons

   Usually we consider the system on the GS when `ekin_conv_thr` $<\sim 10^{-5}$. You could check the value of the fictitious kinetic energy on the standard output (column EKINC).

   Different strategies are available to minimize electrons, but the most used ones are:

   - steepest descent:

     ```
     electron_dynamics = 'sd'
     ```

   - damped dynamics:

     ```
     electron_dynamics = 'damp',
     electron_damping = 0.1,
     ```

     See input description to compute damping factor, usually the value is between 0.1 and 0.5.

4. Once your system is in the GS, depending on how you have prepared the starting atomic configuration, you should do several things:

- if you have set the atomic positions "by hand" and/or from a classical code, check the forces on atoms, and if they are large ($\sim 0.1 - 1.0$ atomic units), you should perform an ionic minimization, otherwise the sistem could break-up during the dynamics.
- if you have taken the positions from a previous run or a previous ab-initio simulation, check the forces, and if they are too small ($\sim 10^{-4}$ atomic units), this means that atoms are already in equilibrium positions and, even if left free, they will not move. Then you need to randomize positions a little bit. see below.

5. Minimize ionic positions.

   As we pointed out in 4) if the interatomic forces are too high, the system could "explode" if we switch on the ionic dynamics. To avoid that we need to relax the system.

   Again there are different strategies to relax the system, but the most used are again steepest descent or damped dynamics for ions and electrons. You could also mix electronic and ionic minimization scheme freely, i.e. ions in steepest and electron in damping or vice versa.

   (a) suppose we want to perform a steepest for ions. Then we should specify the following section for ions:

   ```
   &ions
      ion_dynamics = 'sd',
   /
   ```

   Change also the ionic masses to accelerate the minimization:

   ```
   ATOMIC_SPECIES
    C 2.0d0 c_blyp_gia.pp
    H 2.00d0 h.ps
   ```

   while leaving unchanged other input parameters.

   Note that if the forces are really high (> 1.0 atomic units), you should always use stepest descent for the first relaxation steps (~ 100).

   (b) as the system approaches the equilibrium positions, the steepest descent scheme slows down, so is better to switch to damped dynamics:

   ```
   &ions
      ion_dynamics = 'damp',
   ```

```
    ion_damping = 0.2,
    ion_velocities = 'zero',
/
```

A value of `ion_damping` between 0.05 and 0.5 is usually used for many systems. It is also better to specify to restart with zero ionic and electronic velocities, since we have changed the masses. Change further the ionic masses to accelerate the minimization:

```
ATOMIC_SPECIES
 C 0.1d0 c_blyp_gia.pp
 H 0.1d0 h.ps
```

(c) when the system is really close to the equilibrium, the damped dynamics slow down too, especially because, since we are moving electron and ions together, the ionic forces are not properly correct, then it is often better to perform a ionic step every N electronic steps, or to move ions only when electron are in their GS (within the chosen threshold).

This can be specified adding, in the ionic section, the `ion_nstepe` parameter, then the ionic input section become as follows:

```
&ions
    ion_dynamics = 'damp',
    ion_damping = 0.2,
    ion_velocities = 'zero',
    ion_nstepe = 10,
/
```

Then we specify in the control input section:

```
    etot_conv_thr = 1.d-6,
    ekin_conv_thr = 1.d-5,
    forc_conv_thr = 1.d-3
```

As a result, the code checks every 10 electronic steps whether the electronic system satisfies the two thresholds `etot_conv_thr`, `ekin_conv_thr`: if it does, the ions are advanced by one step. The process thus continues until the forces become smaller than `forc_conv_thr`.

Note that to fully relax the system you need many run, and different strategies, that you shold mix and change in order to speed-up the convergence. The process is not automatic, but is strongly based on experience, and trial and error.

Remember also that the convergence to the equilibrium positions depends on the energy threshold for the electronic GS, in fact correct forces (required to move ions toward the minimum) are obtained only when electrons are in their GS. Then a small threshold on forces could not be satisfied, if you do not require an even smaller threshold on total energy.

6. randomization of positions.

   If you have relaxed the system or if the starting system is already in the equilibrium positions, then you need to move ions from the equilibrium positions, otherwise they won't move in a dynamics simulation. After the randomization you should bring electrons on the GS again, in order to start a dynamic with the correct forces and with electrons in the GS. Then you should switch off the ionic dynamics and activate the randomization for each species, specifying the amplitude of the randomization itself. This could be done with the following ionic input section:

   ```
   &ions
      ion_dynamics = 'none',
      tranp(1) = .TRUE.,
      tranp(2) = .TRUE.,
      amprp(1) = 0.01
      amprp(2) = 0.01
   /
   ```

   In this way a random displacement (of max 0.01 a.u.) is added to atoms of specie 1 and 2. All other input parameters could remain the same.

   Note that the difference in the total energy (`etot`) between relaxed and randomized positions can be used to estimate the temperature that will be reached by the system. In fact, starting with zero ionic velocities, all the difference is potential energy, but in a dynamics simulation, the energy will be equipartitioned between kinetic and potential, then to estimate the temperature take the difference in energy (de), convert it in Kelvins, divide for the number of atoms and multiply by 2/3.

   Randomization could be useful also while we are relaxing the system, especially when we suspect that the ions are in a local minimum or in an energy plateau.

7. Start the Car-Parrinello dynamics.

At this point after having minimized the electrons, and with ions displaced from their equilibrium positions, we are ready to start a CP dynamics. We need to specify `'verlet'` both in ionic and electronic dynamics. The threshold in control input section will be ignored, like any parameter related to minimization strategy. The first time we perform a CP run after a minimization, it is always better to put velocities equal to zero, unless we have velocities, from a previous simulation, to specify in the input file. Restore the proper masses for the ions. In this way we will sample the microcanonical ensemble. The input section changes as follow:

```
 &electrons
    emass = 400.d0,
    emass_cutoff = 2.5d0,
    electron_dynamics = 'verlet',
    electron_velocities = 'zero',
 /
 &ions
    ion_dynamics = 'verlet',
    ion_velocities = 'zero',
 /
ATOMIC_SPECIES
C 12.0d0 c_blyp_gia.pp
H 1.00d0 h.ps
```

If you want to specify the initial velocities for ions, you have to set `ion_velocities = 'from_input'`, and add the `IONIC_VELOCITIES` card, with the list of velocities in atomic units.

IMPORTANT: in restarting the dynamics after the first CP run, remember to remove or comment the velocities parameters:

```
 &electrons
    emass = 400.d0,
    emass_cutoff = 2.5d0,
    electron_dynamics = 'verlet',
    ! electron_velocities = 'zero',
 /
 &ions
    ion_dynamics = 'verlet',
    ! ion_velocities = 'zero',
 /
```

otherwise you will quench the system interrupting the sampling of the microcanonical ensemble.

8. Changing the temperature of the system.

   It is possible to change the temperature of the system or to sample the canonical ensemble fixing the average temperature, this is done using the Nosè thermostat. To activate this thermostat for ions you have to specify in the ions input section:

```
&ions
   ion_dynamics = 'verlet',
   ion_temperature = 'nose',
   fnosep = 60.0,
   tempw  = 300.0,
   ! ion_velocities = 'zero',
/
```

   where `fnosep` is the frequency of the thermostat in THz, this should be chosen to be comparable with the center of the vibrational spectrum of the system, in order to excite as many vibrational modes as possible. `tempw` is the desired average temperature in Kelvin.

   It is possible to specify also the thermostat for the electrons, this is usually activated in metal or in system where we have a transfer of energy between ionic and electronic degrees of freedom.

7 Performance issues (PWscf)

7.1 CPU time requirements

The following holds for code `pw.x` and for non-US PPs. For US PPs there are additional terms to be calculated. For phonon calculations, each of the $3N_{at}$ modes requires a CPU time of the same order of that required by a self-consistent calculation in the same system.

The computer time required for the self-consistent solution at fixed ionic positions, T_{scf}, is:

$$T_{scf} = N_{iter} \cdot T_{iter} + T_{init}$$

where N_{iter} = `niter` = number of self-consistency iterations, T_{iter} = CPU time for a single iteration, T_{sub} = initialization time for a single iteration. Usually $T_{init} << N_{iter} \cdot T_{iter}$.

The time required for a single self-consistency iteration T_{iter} is:

$$T_{iter} = N_k \cdot T_{diag} + T_{rho} + T_{scf}$$

where N_k = number of k-points, T_{diag} = CPU time per hamiltonian iterative diagonalization, T_{rho} = CPU time for charge density calculation, T_{scf} = CPU time for Hartree and exchange-correlation potential calculation.

The time for a Hamiltonian iterative diagonalization T_{diag} is:

$$T_{diag} = N_h \cdot T_h + T_{orth} + T_{sub}$$

where N_h = number of $H\psi$ products needed by iterative diagonalization, T_h = CPU time per $H\psi$ product, T_{orth} = CPU time for orthonormalization, T_{sub} = CPU time for subspace diagonalization.

The time T_h required for a $H\psi$ product is

$$T_h = a_1 \cdot M \cdot N + a_2 \cdot M \cdot N_1 \cdot N_2 \cdot N_3 \cdot \log(N_1 \cdot N_2 \cdot N_3) + a_3 \cdot M \cdot P \cdot N.$$

The first term comes from the kinetic term and is usually much smaller than the others. The second and third terms come respectively from local and nonlocal potential. a_1, a_2, a_3 are prefactors, M = number of valence bands, N = number of plane waves (basis set dimension), N_1, N_2, N_3 = dimensions of the FFT grid for wavefunctions ($N_1 \cdot N_2 \cdot N_3 \sim 8N$), P = number of projectors for PPs (summed on all atoms, on all values of the angular momentum l, and $m = 1, \ldots, 2l+1$)

The time T_{orth} required by orthonormalization is

$$T_{orth} = b_1 * M_x^2 * N$$

and the time T_{sub} required by subspace diagonalization is

$$T_{sub} = b_2 * M_x^3$$

where b_1 and b_2 are prefactors, M_x = number of trial wavefunctions (this will vary between M and a few times M, depending on the algorithm).

The time T_{rho} for the calculation of charge density from wavefunctions is

$$T_{rho} = c_1 \cdot M \cdot Nr_1 \cdot Nr_2 \cdot Nr_3 \cdot \log(Nr_1 \cdot Nr_2 \cdot Nr_3) + c_2 \cdot M \cdot Nr_1 \cdot Nr_2 \cdot Nr_3 + T_{us}$$

where c_1, c_2, c_3 are prefactors, Nr_1, Nr_2, Nr_3 = dimensions of the FFT grid for charge density ($Nr_1 \cdot Nr_2 \cdot Nr_3 \sim 8N_g$, where N_g = number of G-vectors for the charge density), and T_{us} = CPU time required by ultrasoft contribution (if any).

The time T_{scf} for calculation of potential from charge density is

$$T_{scf} = d_2 \cdot Nr_1 \cdot Nr_2 \cdot Nr_3 + d_3 \cdot Nr_1 \cdot Nr_2 \cdot Nr_3 \cdot \log(Nr_1 \cdot Nr_2 \cdot Nr_3)$$

where d_1, d_2 are prefactors.

7.2 Memory requirements

A typical self-consistency or molecular-dynamics run requires a maximum memory in the order of O double precision complex numbers, where

$$O = m \cdot M \cdot N + P \cdot N + p \cdot N_1 \cdot N_2 \cdot N_3 + q \cdot Nr_1 \cdot Nr_2 \cdot Nr_3$$

with m, p, q = small factors; all other variables have the same meaning as above. Note that if the Γ-point only ($\mathbf{q} = 0$) is used to sample the Brillouin Zone, the value of N will be cut into half.

Code `memory.x` yields a rough estimate of the memory required by `pw.x` and checks for the validity of the input data file as well. Use it exactly as `pw.x`.

The memory required by the phonon code follows the same patterns, with somewhat larger factors m, p, q.

7.3 File space requirements

A typical `pw.x` run will require an amount of temporary disk space in the order of O double precision complex numbers:

$$O = N_k \cdot M \cdot N + q \cdot Nr_1 \cdot Nr_2 \cdot Nr_3$$

where $q = 2 \cdot$ `mixing_ndim` (number of iterations used in self-consistency, default value = 8) if `disk_io` is set to `'high'` or not specified; $q = 0$ if `disk_io='low'` or `'minimal'`.

7.4 Parallelization issues

`pw.x` can run in principle on any number of processors (up to `maxproc`, presently fixed at 128 in `PW/para.f90`). The N_p processors can be divided into N_{pk} pools of N_{pr} processors, $N_p = N_{pk} * N_{pr}$. The k-points are divided across N_{pk} pools ("k-point parallelization"), while both R- and G-space grids are divided across the N_{pr} processors of each pool ("PW parallelization"). A third level of parallelization, on the number of bands, is currently confined to the calculation of a few quantities that would not be parallelized at all otherwise. A fourth level of parallelization, on the number of NEB images, is available for NEB calculation only.

The effectiveness of parallelization depends on the size and type of the system and on a judicious choice of the N_{pk} and N_{pr}:

- k-point parallelization is very effective if N_{pk} is a divisor of the number of k-points (linear speedup guaranteed), *but* it does not reduce the amount of memory per processor taken by the calculation. As a consequence, large systems may not fit into memory. The same applies to parallelization over NEB images.

- PW parallelization works well if N_{pr} is a divisor of both dimensions along the z axis of the FFT grids, N_3 and Nr_3 (which may coincide). It does not scale so well as k-point parallelization, but it reduces both CPU time AND memory (the latter almost linearly).

- Optimal serial performances are achieved when the data are as much as possible kept into the cache. As a side effect, one can achieve better than linear scaling with the number of processors, thanks to the increase in serial speed coming from the reduction of data size (making it easier for the machine to keep data in the cache).

Note that for each system there is an optimal range of number of processors on which to run the job. A too large number of processors will yield performance degradation, or may cause the parallelization algorithm to fail in distributing properly R- and G-space grids.

Note also that Beowulf-style machines (PC clusters) may have disappointing parallelization performances unless they have a decent communication hardware (at least Gigabit ethernet). Do not expect good scaling with cheap hardware: plane-wave calculations are not at all an "embarrassing parallel" problem. Note that multiprocessor motherboards for Intel Pentium CPUs typically have just one memory bus for all processors. This dramatically slows down any code doing massive access to memory (as most codes in

the Quantum-ESPRESSO package do) that runs on processors of the same motherboard.

8 Troubleshooting (PWscf)

Almost all problems in PWscf arise from incorrect input data and result in error stops. Error messages should be self-explanatory, but unfortunately this is not always true. If the code issues a warning messages and continues, pay attention to it but do not assume that something is necessarily wrong in your calculation: most warning messages signal harmless problems.

Note for PC Linux clusters in parallel execution: in at least some versions of MPICH, the current directory is set to the directory where the *executable code* resides, instead of being set to the directory where the code is executed. This MPICH weirdness may cause unexpected failures in some postprocessing codes that expect a data file in the current directory. Workaround: use symbolic links, or copy the executable to the current directory.

Typical `pw.x` and/or `ph.x` (mis-)behavior:

`pw.x` yields a message like "error while loading shared libraries: ... cannot open shared object file" and does not start. Possible reasons:

- If you are running on the same machines on which the code was compiled, this is a library configuration problem. The solution is machine-dependent. On Linux, find the path to the missing libraries; then either add it to file `/etc/ld.so.conf` and run `ldconfig` (must be done as root), or add it to variable `LD_LIBRARY_PATH` and export it. Another possibility is to load non-shared version of libraries (ending with `.a`) instead of shared ones (ending with `.so`).
- If you are *not* running on the same machines on which the code was compiled: you need either to have the same shared libraries installed on both machines, or to load statically all libraries (using appropriate `configure` or loader options). The same applies to Beowulf-style parallel machines: the needed shared libraries must be present on all PC's.

errors in examples with parallel execution If you get error messages in the example scripts – i.e. not errors in the codes – on a parallel machine, such as e.g. : "`run_example: -n: command not found`" you have forgotten the "`" in the definitions of `PARA_PREFIX` and `PARA_POSTFIX`.

`pw.x` prints the first few lines and then nothing happens (parallel execution). If the code looks like it is not reading from input, maybe it isn't: the MPI libraries need to be properly configured to accept input

redirection. See section "Running on parallel machines", or inquire with your local computer wizard (if any).

`pw.x` stops with error in reading. There is an error in the input data. Usually it is a misspelled namelist variable, or an empty input file. Note that out-of-bound indices in dimensioned variables read in the namelist may cause the code to crash with really mysterious error messages. Also note that input data files containing `^M` (Control-M) characters at the end of lines (typically, files coming from Windows PC) may yield error in reading. If none of the above applies and the code stops at the first namelist ("control") and you are running in parallel: your MPI libraries might not be properly configured to allow input redirection, so that what you are effectively reading is an empty file. See section "Running on parallel machines", or inquire with your local computer wizard (if any).

`pw.x` mumbles something like "cannot recover" or "error reading recover file". You are trying to restart from a previous job that either produced corrupted files, or did not do what you think it did. No luck: you have to restart from scratch.

`pw.x` stops with error in cdiagh or cdiaghg. Possible reasons:

- serious error in data, such as bad atomic positions or bad crystal structure/supercell;
- a bad PP (for instance, with a ghost);
- a failure of the algorithm performing subspace diagonalization. The LAPACK algorithms used by cdiagh or cdiaghg are very robust and extensively tested. Still, it may seldom happen that such algorithms fail. In at least one case the failures was tracked to the non-positiveness of the S matrix appearing in the US-PP formalism. In other cases, the error is found to be non reproducible on different architectures and disappearing if the calculation is repeated with even minimal changes in its parameters. In both cases, the reasons for such behavior are unclear and the only advice is to use conjugate-gradient diagonalization (`diagonalization='cg'`), a slower but very robust algorithm, and see what happens.
- HP-Compaq alphas with `cxml` libraries: try to use compiled BLAS and LAPACK (or better, ATLAS) instead of those contained in `cxml` (just load them before `cxml`).

`pw.x` **crashes with "floating invalid" or "floating divide by zero".** If this happens on HP-Compaq True64 Alpha machines with an old version of the compiler: the compiler is most likely buggy. Otherwise, move to next item.

`pw.x` **crashes with no error message at all.** This happens quite often in parallel execution, or under a batch queue, or if you are writing the output to a file. When the program crashes, part of the output, including the error message, may be lost, or hidden into error files where nobody looks into. It is the fault of the operating system, not of the code. Try to run interactively and to write to the screen. If this doesn't help, move to next point.

`pw.x` **crashes with "segmentation fault" or similarly obscure messages.** Possible reasons:

- too much RAM memory requested (see next item).
- if you are using highly optimized mathematical libraries, verify that they are designed for your hardware. In particular, for Intel compiler and MKL libraries, verify that you loaded the correct set of CPU-specific MKL libraries.
- buggy compiler. If you are using Portland or Intel compilers on Linux PC's or clusters, see section 2.4, "Installation issues".

`pw.x` **works for simple systems, but not for large systems or whenever more RAM is needed.** Possible solutions:

- increase the amount of RAM you are authorized to use (which may be much smaller than the available RAM). Ask your system administrator if you don't know what to do.
- reduce `nbnd` to the strict minimum, or reduce the cutoffs, or the cell size.
- use conjugate-gradient (`diagonalization='cg'`: slow but very robust) or DIIS (`diagonalization='diis'`: fast but not very robust): both requires less memory than the default Davidson algorithm.
- in parallel execution, use more processors, or use the same number of processors with less pools. Remember that parallelization with respect to k-points (pools) does not distribute memory: parallelization with respect to **R**- (and **G**-) space does.

- IBM only (32-bit machines): if you need more than 256 MB you must specify it at link time (option `-bmaxdata`).
- buggy or weird-behaving compiler. Some versions of the Portland and Intel compilers on Linux PC's or clusters have this problem. For Intel ifort 8.1, the problem seems to be due to the allocation of large automatic arrays that exceeds the available stack. Increasing the stack size (with commands `limits` or `ulimit`) may solve the problem.

`pw.x` crashes in parallel execution with an obscure message related to MPI errors. With LAM-MPI, add `-D__LAM` to preprocessing options in `make.sys` and recompile. See info from Axel Kohlmeyer:
`http://www.democritos.it/pipermail/pw_forum/2005-April/002338.html`

`pw.x` runs but nothing happens. Possible reasons:

- in parallel execution, the code died on just one processor. Unpredictable behavior may follow.
- in serial execution, the code encountered a floating-point error and goes on producing NaN's (Not a Number) forever unless exception handling is on (and usually it isn't). In both cases, look for one of the reasons given above.
- maybe your calculation will take more time than you expect.

`pw.x` yields weird results. Possible solutions:

- if this happen after a change in the code or in compilation or preprocessing options, try `make clean` and recompile. The `make` command should take care of all dependencies, but do not rely too heavily on it. If the problem persists, `make clean` and recompile with reduced optimization level.
- maybe your input data are weird.

`pw.x` stops with error message "the system is metallic, specify occupations". You did not specify state occupations, but you need to, since your system appears to have an odd number of electrons. The variable controlling how metallicity is treated is `occupations` in namelist `&SYSTEM`. The default, `occupations='fixed'`, occupies the lowest `nelec/2` states and works only for insulators with a gap. In all other cases, use `'smearing'` or `'tetrahedra'`. See file `INPUT_PW` for more details.

`pw.x` stops with "unexpected error" in `efermi`. Possible reasons:

- serious error in data, such as bad number of electrons, insufficient number of bands, absurd value of broadening, or too few tetrahedra;
- the Fermi energy is found by bisection assuming that the integrated DOS $N(E)$ is an increasing function of the energy. This is *not* guaranteed for Methfessel-Paxton smearing of order 1 and can give problems when very few k-points are used. Use some other smearing function: simple Gaussian broadening or, better, Marzari-Vanderbilt "cold smearing".

in parallel execution, `pw.x` stops complaining that "some processors have no planes" or "smooth planes" or some other strange error. Your system does not require that many processors: reduce the number of processors to a more sensible value. In particular, both N_3 and Nr_3 must be $\geq N_{pr}$ (see section 7, "Performance Issues", and in particular section 7.4, "Parallelization issues", for the meaning of these variables).

the FFT grids in `pw.x` are machine-dependent. Yes, they are! The code automatically chooses the smallest grid that is compatible with the specified cutoff in the specified cell, *and* is an allowed value for the FFT library used. Most FFT libraries are implemented, or perform well, only with dimensions that factors into products of small numers (2, 3, 5 typically, sometimes 7 and 11). Different FFT libraries follow different rules and thus different dimensions can result for the same system on different machines (or even on the same machine, with a different FFT). See function `allowed` in `Modules/fft_scalar.f90`.

As a consequence, the energy may be slightly different on different machines. The only piece that depends explicitely on the grid parameters is the XC part of the energy that is computed numerically on the grid. The differences should be small, though, expecially for LDA calculations.

Manually setting the FFT grids to a desired value is possible, but slightly tricky, using input variables `nr1, nr2, nr3` and `nr1s, nr2s, nr3s`. The code will still increase them if not acceptable. Automatic FFT grid dimensions are slightly overestimated, so one may try — very carefully — to reduce them a little bit. The code will stop if too small values are required, it will waste CPU time and memory for too large values.

Note that in parallel execution, it is very convenient to have FFT grid dimensions along z that are a multiple of the number of processors.

"warning: symmetry operation # N not allowed". This is not an error. `pw.x` determines first the symmetry operations (rotations) of the Bravais lattice; then checks which of these are symmetry operations of the system (including if needed fractional translations). This is done by rotating (and translating if needed) the atoms in the unit cell and verifying if the rotated unit cell coincides with the original one.

If a symmetry operation contains a fractional translation that is incompatible with the FFT grid, it is discarded in order to prevent problems with symmetrization. Typical fractional translations are 1/2 or 1/3 of a lattice vector. If the FFT grid dimension along that direction is not divisible respectively by 2 or by 3, the symmetry operation will not transform the FFT grid into itself.

`pw.x` doesn't find all the symmetries you expected. See above to learn how PWscf finds symmetry operations. Some of them might be missing because:

- the number of significant figures in the atomic positions is not large enough. In file `PW/eqvect.f90`, the variable `accep` is used to decide whether a rotation is a symmetry operation. Its current value (10^{-5}) is quite strict: a rotated atom must coincide with another atom to 5 significant digits. You may change the value of `accep` and recompile.

- they are not acceptable symmetry operations of the Bravais lattice. This is the case for C_{60}, for instance: the I_h icosahedral group of C_{60} contains 5-fold rotations that are incompatible with translation symmetry.

- the system is rotated with respect to symmetry axis. For instance: a C_{60} molecule in the fcc lattice will have 24 symmetry operations (T_h group) only if the double bond is aligned along one of the crystal axis; if C_{60} is rotated in some arbitrary way, `pw.x` may not find any symmetry, apart from inversion.

- they contain a fractional translation that is incompatible with the FFT grid (see previous paragraph). Note that if you change cutoff or unit cell volume, the automatically computed FFT grid changes, and this may explain changes in symmetry (and in the number of k-points as a consequence) for no apparent good reason (only if you have fractional translations in the system, though).

- a fractional translation, without rotation, is a symmetry operation of the system. This means that the cell is actually a supercell. In this

case, all symmetry operations containing fractional translations are disabled. The reason is that in this rather exotic case there is no simple way to select those symmetry operations forming a true group, in the mathematical sense of the term.

the CPU time is time-dependent! Yes it is! On most machines and on most operating systems, depending on machine load, on communication load (for parallel machines), on various other factors (including maybe the phase of the moon), reported CPU times may vary quite a lot for the same job. Also note that what is printed is supposed to be the CPU time per process, but with some compilers it is actually the wall time.

"warning : N eigenvectors not converged ..." This is a warning message that can be safely ignored if it is not present in the last steps of self-consistency. If it is still present in the last steps of self-consistency, and if the number of unconverged eigevector is a significant part of the total, it may signal serious trouble in self-consistency (see next point) or something badly wrong in input data.

"warning : negative or imaginary charge...", or "...core charge ...", or "npt with rhoup<0..." or "rhodw<0..." These are warning messages that can be safely ignored unless the negative or imaginary charge is sizable, let us say O(0.1). If it is, something seriously wrong is going on. Otherwise, the origin of the negative charge is the following. When one transforms a positive function in real space to Fourier space and truncates at some finite cutoff, the positive function is no longer guaranteed to be positive when transformed back to real space. This happens only with core corrections and with ultrasoft pseudopotentials. In some cases it may be a source of trouble (see next point) but it is usually solved by increasing the cutoff for the charge density.

self-consistency is slow or does not converge. Reduce `mixing_beta` from the default value (0.7) to $\sim 0.3 - 0.1$ or smaller, or try a different `mixing_mode`. You may also try to increase `mixing_ndim` to more than 8 (default value). Beware: the larger `mixing_ndim`, the larger the amount of memory you need.

If the above doesn't help: verify if your system is metallic or is close to a metallic state, especially if you have few k-points. If the highest occupied and lowest unoccupied state(s) keep exchanging place during self-consistency, forget about reaching convergence. A typical sign of such behavior is that

the self-consistency error goes down, down, down, than all of a sudden up again, and so on. Usually one can solve the problem by adding a few empty bands and a broadening.

Specific to US PP: the presence of negative charge density regions due to either the pseudization procedure of the augmentation part or to truncation at finite cutoff may give convergence problems. Raising the `ecutrho` cutoff for charge density will usually help, especially in gradient-corrected calculations.

structural optimization is slow or does not converge. Typical structural optimizations, based on the BFGS algorithm, converge to the default thresholds (`etot_conv_thr` and `forc_conv_thr`) in 15-25 BFGS steps (depending on the starting configuration). This may not happen when your system is characterized by "floppy" low-energy modes, that make very difficult — and of little use anyway — to reach a well converged structure, no matter what. Other possible reasons for a problematic convergence are listed below.

Close to convergence the self-consistency error in forces may become large with respect to the value of forces. The resulting mismatch between forces and energies may confuse the line minimization algorithm, which assumes consistency between the two. The code reduces the starting self-consistency threshold `conv_thr` when approaching the minimum energy configuration, up to a factor defined by `upscale`. Reducing `conv_thr` (or increasing `upscale`) yields a smoother structural optimization, but if `conv_thr` becomes too small, electronic self-consistency may not converge. You may also increase variables `etot_conv_thr` and `forc_conv_thr` that determine the threshold for convergence (the default values are quite strict).

A limitation to the accuracy of forces comes from the absence of perfect translational invariance. If we had only the Hartree potential, our PW calculation would be translationally invariant to machine precision. The presence of an exchange-correlation potential introduces Fourier components in the potential that are not in our basis set. This loss of precision (more serious for gradient-corrected functionals) translates into a slight but detectable loss of translational invariance (the energy changes if all atoms are displaced by the same quantity, not commensurate with the FFT grid). This sets a limit to the accuracy of forces. The situation improves somewhat by increasing the `ecutrho` cutoff.

`ph.x` stops with "error reading file". The data file produced by `pw.x` is bad or incomplete or produced by an incompatible version of the code. In parallel execution: if you did not set `wf_collect=.true.`, the number

of processors and pools for the phonon run should be the same as for the self-consistent run; all files must be visible to all processors.

`ph.x` mumbles something like "cannot recover" or "error reading recover file". You have a bad restart file from a preceding failed execution. Remove all files `recover*` in `outdir`.

`ph.x` says "occupation numbers probably wrong" and continues; or "phonon + tetrahedra not implemented" and stops You have a metallic or spin-polarized system but occupations are not set to "smearing". Note that the correct way to calculate occupancies must be specified in the input data of the non-selfconsistent calculation, if the phonon code reads data from it. The non-selfconsistent calculation will not use this information but the phonon code will.

`ph.x` does not yield acoustic modes with $\omega = 0$ at $\mathbf{q} = 0$. This may not be an error: the Acoustic Sum Rule (ASR) is never exactly verified, because the system is never exactly translationally invariant as it should be (see the discussion above). The calculated frequency of the acoustic mode is typically less than 10 cm^{-1}, but in some cases it may be much higher, up to 100 cm^{-1}. The ultimate test is to diagonalize the dynamical matrix with program `dynmat.x`, imposing the ASR. If you obtain an acoustic mode with a much smaller ω (let's say $< 1\mathrm{cm}^{-1}$) with all other modes virtually unchanged, you can trust your results.

`ph.x` yields really lousy phonons, with bad or negative frequencies or wrong symmetries or gross ASR violations. Possible reasons:

- wrong data file file read.
- wrong atomic masses given in input will yield wrong frequencies (but the content of file `fildyn` should be valid, since the force constants, not the dynamical matrix, are written to file).
- convergence threshold for either SCF (`conv_thr`) or phonon calculation (`tr2_ph`) too large (try to reduce them).
- maybe your system *does* have negative or strange phonon frequencies, with the approximations you used. A negative frequency signals a mechanical instability of the chosen structure. Check that the structure is reasonable, and check the following parameters:

- The cutoff for wavefunctions, `ecutwfc`
- For US PP: the cutoff for the charge density, `ecutrho`
- The k-point grid, especially for metallic systems!

"Wrong degeneracy" error in star_q. Verify the **q**-point for which you are calculating phonons. In order to check whether a symmetry operation belongs to the small group of **q**, the code compares **q** and the rotated **q**, with an acceptance tolerance of 10^{-5} (set in routine `PW/eqvect.f90`). You may run into trouble if your **q**-point differs from a high-symmetry point by an amount in that order of magnitude.